Commensense Knowledge Base Completion [LTTG16]

Xiang Li, Aynaz Taheri, Lifu Tu, Kevin Gimpel

Presenter: Xuefeng Hu

03/04/2019
Outline

Introduction

Models

Training

Evaluation and Results

Discussion
Introduction: Commonsense Knowledge

Figure 1: A high-level view of the knowledge ConceptNet has about a cluster of related concepts.

**Figure 1:** ConceptNet[SH12] example
Introduction: Problem Statement

- **Ultimate Goal**: To build a Knowledge Base for commonsense knowledge.
- **What we have**: Manually labeled curated resources, such as ConceptNet[SH12].
- **What needs to be done**: Grow the Commonsense Knowledge Base by automatically generate/detect unlabeled knowledge, with the help of the curated resource.

Figure 2: The Problem of formulating Commonsense Knowledge Base
Introduction: Goal

- **Formulation**: a knowledge entry is defined by a tuple $< t_1, R, t_2 >$, where $t_1, t_2$ can be any phrases, with no restrictions, and $R$ is from a set of pre-defined relations.

- **Goal**: the goal is to learn function $score(t_1, R, t_2)$ that predict high score on reasonable knowledge and low score on less-likely knowledge.
Models: From phrases to vectors

- **Word Embedding**
  - Word2Vec\[MCCD13\] trained with Open Mind Common Sense (OMCS, [SH12]) sentences that contains the training tuple phrases, with inserted relations:

    \(< \text{soak in a hotspring, CAUSES, get pruny skin} >\)
    \(\Rightarrow\) The effect of [soaking in a hotspring] is [getting pruny skin]
    \(\Rightarrow\) The effect of soak in a hotspring \textit{CAUSES} get pruny skin

- Pre-trained Word Embedding: GloVe\[PSM14\].
- Pre-trained Word Embedding: Paragram-SimLex\[WBGL15a\].

- **Merge**
  - **AVG**: just averaging over all the words embedding from the same phrase into a single embedding.
  - **LSTM**: use a LSTM to merge the embedding of multiple words into a single embedding.
Models: Bilinear Models

- **Length Control:**
  \[ u_i = a(Wv_i + b) \]
  which transform the merged phrase embedding \( v_i \) into vector \( u_i \in \mathbb{R}^N \) with a fully connected layer with activation.

- **Score function:**
  \[ score_{bilinear} = u_1^T M_R u_2 \]
  where \( M_R \) is learnable \( N \times N \) matrix for relation \( R \)
Models: Bilinear Models

Figure 3: The flow chart of the Bilinear models
Models: Neural Network Model: DNN-AVG

Figure 4: Neural Network Model with averaged embedding
Models: Neural Network Model: DNN-LSTM

Figure 5: Neural Network Model with LSTM generated embedding
Training: Negative Example Generation

For each positive sample $\tau = < t_1, R, t_2 >$, 3 negative sample $\tau_{neg}(t_1), \tau_{neg}(t_2), \tau_{neg}(R)$ can be generated by replacing one of the elements in tuple. There are 3 ways to replace the elements:

- **Random Sampling**: replace by its counterpart in a randomly-chosen tuple in the same batch.

- **Max Sampling**: Take $t_1$ as example. Replace $t_1$ by $t_1'$ such that $t_1'$ is the most unlikely (with respect to the current score function) term appear in tuple $< t, R, t_2 >$:

  $$t_1' = \arg\max_{t: < t, R', t_2 '> \in \mu \setminus \tau} \text{score}(t, R, t_2)$$

- **Mix Sampling**: 50% random sampling + 50% max sampling.
Training: Loss Functions

- **Hinge Loss**: the hinge loss is defined upon a set of a positive sample and its 3 negative samples.

\[
loss_{\text{hinge}}(\tau) = \max\{0, \gamma - \text{score}(\tau) + \text{score}(\tau_{\text{neg}}(t_1))\}
\]
\[
+ \max\{0, \gamma - \text{score}(\tau) + \text{score}(\tau_{\text{neg}}(t_2))\}
\]
\[
+ \max\{0, \gamma - \text{score}(\tau) + \text{score}(\tau_{\text{neg}}(R))\}
\]

where \(\gamma\) is the margin gap.

- **Binary Cross-Entropy**: the Cross Entropy loss is defined for every sample, with its label \(y \in \{0, 1\}\):

\[
loss_{\text{CE}}(\tau, y) = -y \log \sigma(\text{score}(\tau)) - (1 - y) \log \sigma(1 - \text{score}(\tau))
\]
Training: Regularization

- For DNN models, the simple $l_2$ loss $\lambda \| \theta \|^2_2$ are used.
- For Bilinear models, a regularization on relation matrix $M_R$ is used along with the $l_2$ regularization:

$$\lambda_1 \| \theta \|^2_2 + \lambda_2 \sum_R \| M_R - I_N \|^2_2$$
Dataset

- ConceptNet5 Dataset [SH12] is used as the dataset for training and testing.
- Sort all tuples in ConceptNet by their confidence score. Pick top 1200 tuples as the TEST set, next 600 tuples as the first development set (DEV1) and another next 600 tuples as the second development set (DEV2). Pick another 100000 tuples from the rest for training.
- For samples $\tau$ in TEST, DEV1, and DEV2, randomly generate a negative sample by swapping one of the components of $\tau$ with another sample $\tau'$ from the same set. Result in a Test set with 2400 tuples, DEV1 with 1200 tuples and DEV2 with 1200 tuples.
- DEV1 is used for choosing the optimal threshold to binarize the 0-1 score, and DEV2 is used for tuning all the other hyper-parameters, including the activation function and pooling ways.
Evaluation: Baselines

- **Similar Fact Count (Count)**: For each test tuple $\tau = < t_1, R, t_2 >$, count the number of tuple $\tau' = < t'_1, R', t'_2 >$ in training set that:
  - $t_1 = t'_1; R = R'; head(t_2) = head(t'_2)$, or
  - $t_2 = t'_2; R = R'; head(t_1) = head(t'_1)$.

  where $head()$ gives the head words of the phrase by [KM03].

- **Argument Similarity (ArgSim)**: Cosine distance between $v_1, v_2$, the averaging embedding for phrases $t_1, t_2$.

- **Max Similarity (MaxSim)**: The maximum similarity between the testing tuple and any tuple in the training set. The similarity is defined by the cosine distance between catenating vector of $t_1, t_2$ and $R$. 
Result: Word Embedding Comparison

<table>
<thead>
<tr>
<th></th>
<th>GloVe</th>
<th>PARAGRAM</th>
<th>CN-trained</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgSim</td>
<td>68</td>
<td>69</td>
<td>73</td>
</tr>
<tr>
<td>MaxSim</td>
<td>73</td>
<td>70</td>
<td>82</td>
</tr>
</tbody>
</table>

Table 2: Accuracies (\%) on DEV2 of two baselines using three different sets of word embeddings. Our ConceptNet-trained embeddings outperform GloVe and PARAGRAM embeddings.

Figure 6: Word Embedding Comparison over baseline methods

- **Count** does not require word embedding, therefore there is not result for it here.
- The experiments on **ArgSim** and **MaxSim** shows that CN-trained (Word2Vec trained with ConceptNet training tuples) largely outperform the other embedding.
- The rest experiments are performed all with CN-trained embedding, if needed.
Result: Sample Generation Comparison I

Table 3: Accuracies (%) on DEV2 of models trained with two loss functions (cross entropy (CE) and hinge) and three sampling strategies (random, mix, and max). The best accuracy for each model is shown in bold. Cross entropy with random sampling is best across models and is also fastest (see Table 4).

Figure 7: Loss function and negative sample formulation compare

Table 4: Loss function runtime comparison (seconds per epoch) of the DNN models.

Figure 8: Run-time analysis over different negative example sampling
The result of four models on DEV2 are shown in the table. From the table we can see

- For all testing methods, the Cross-Entropy Loss has higher performance.
- For all testing methods, the random negative example sampling has higher performance.
- Since random sampling does not need to go over all tuples in training set, it takes a shorter time than other methods.
- The low performance of Max Sampling is surprising, contradicts the result from [WBGL15b]. The Authors found that in training, the Max Sampling often forms the True tuples, and argue that it caused by the characteristics of the ConceptNet.
### Table 5: Accuracies (%) of baselines and final model configurations on DEV2 and TEST.

“+ data” uses enlarged training set of size 300,000, and then doubles this training set by including tuples with conjugated forms; see text for details. Human performance on DEV2 was estimated from a sample of size 100.

<table>
<thead>
<tr>
<th>Model Configuration</th>
<th>DEV2</th>
<th>TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>75.4</td>
<td>79.0</td>
</tr>
<tr>
<td>ArgSim</td>
<td>72.9</td>
<td>74.2</td>
</tr>
<tr>
<td>MaxSim</td>
<td>81.9</td>
<td>83.5</td>
</tr>
<tr>
<td>Bilinear AVG</td>
<td>90.3</td>
<td>91.7</td>
</tr>
<tr>
<td>Bilinear LSTM</td>
<td>90.8</td>
<td>90.7</td>
</tr>
<tr>
<td>DNN AVG</td>
<td>91.3</td>
<td>92.0</td>
</tr>
<tr>
<td>DNN LSTM</td>
<td>88.1</td>
<td>89.2</td>
</tr>
<tr>
<td>Bilinear AVG + data</td>
<td>91.8</td>
<td>92.5</td>
</tr>
<tr>
<td>human</td>
<td>∼95.0</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 9:** Final Results on DEV2 and TEST
The table shows that for both Bilinear and DNN model, the AVG merging outperforms the LSTM merging. The possible explanation provided by the author is that most phrases contain only 2-3 words, which are too short for the LSTM architecture.

All 4 models easily outperforms the baselines. In particular, the Bilinear AVG and DNN AVG are having the best performance.

An additional experiment is performed with the Bilinear AVG method, by adding tuples formed by unconjugated phrases for all training sample:

\[
\text{< soak in a hotspring, CAUSES, get pruny skin >}
\]

\[\Rightarrow\text{The effect of [soaking in a hotspring] is [getting pruny skin]}
\]

\[\Rightarrow\text{< soaking in a hotspring, CAUSES, getting pruny skin >}
\]
In this section, the author evaluates the model’s ability to score novel tuples generated from ConceptNet and Wikipedia.

- For ConceptNet, prepare tuples using random sampling as preparing the negative samples.

- For Wikipedia, first run the Stanford part-of-speech (POS) tagger [TKMS03] on the terms in ConceptNet training tuples, and retain 50 most frequent term pair tag sequences for each relation, and retain 15 POS tag sequences for each relation, with heuristic filtering to ensure non-trivial relations. Then the novel tuple on Wikipedia can be generated by extracting word sequence pairs corresponding to the relation POS tag sequence pairs, with the removal of unsuitable tuples (such as inefficient gap).
Novel Tuple Evaluation: Evaluation

- To evaluate the method on the novel tuples: manual analysis of a tuple with high predicted score, using a 0-4 scoring metric: 0-Nonsense; 1-False; 2-Not Sure; 3-Sometimes True; 4-Generally True.

<table>
<thead>
<tr>
<th>tuples</th>
<th>quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>high-confidence CN tuples</td>
<td>3.68</td>
</tr>
<tr>
<td>medium-confidence CN tuples</td>
<td>3.14</td>
</tr>
<tr>
<td>novel CN tuples, ranked by MaxSim</td>
<td>2.74</td>
</tr>
<tr>
<td>novel CN tuples, ranked by Bilinear AVG</td>
<td>3.20</td>
</tr>
<tr>
<td>novel Wiki tuples, ranked by Bilinear AVG</td>
<td>2.78</td>
</tr>
</tbody>
</table>

Table 7: Average quality scores from manual evaluation of novel tuples. Each row corresponds to a different set of tuples. See text for details.

Figure 10: Evaluation of MaxSim and Bilinear AVG in a 0-4 Manual Scoring
Table 8: Top ranked tuples extracted from two example sentences and scored by Bilinear AVG model. * = contained in ConceptNet.
Conclusion

- Proposed different methods of scoring tuple, based on training tuples from ConceptNet.
- Experiments on different settings in Word Embedding (CN-trained and pretrained embeddings), Phrase-based feature generation (AVG and LSTM), Model Architecture (Bilinear and DNN), Loss Functions (Hinge and CE) and Negative Example Sampling (Random and MAX).
- Generation and Evaluation on novel generated tuples on ConceptNet and Wikipedia.


References II


References III
